Ubuntu Command Line Quickstart

By Andrew Hudson,Matthew Helmke,Paul Hudson,Ryan Troy

Date: Feb 3, 2011

Sample Chapter is provided courtesy of Sams Publishing.

Return to the article

This chapter looks at some of the basic commands that you need to know to be productive at the Ubuntu command line. You will find out how to get to the command line, and learn some of the commands used to navigate the file system.

In This Chapter

The Linux command line, sometimes called the terminal, is one of the most powerful tools available for computer system administration and maintenance. It is an efficient way to perform complex tasks accurately and much more easily than it would seem at a first glance. Knowledge of the commands available to you and also how to string them together will make using Ubuntu even easier for many tasks. Many of the commands were created by the GNU Project as free software analogs to previously existing proprietary UNIX commands. If you are interested, you can learn more about the GNU Project at www.gnu.org/gnu/thegnuproject.html.

This chapter looks at some of the basic commands that you need to know to be productive at the command line. You will find out how to get to the command line, and learn some of the commands used to navigate the file system. Later in the book we present a Command Line Masterclass (Chapter 30), which explores the subject in greater depth. The skills you learn in this chapter will help you get started using the command line with confidence.

What Is the Command Line?

If you spend any amount of time with experienced Linux users, you will hear them mention the command line. Some, especially those who have begun their journey in the Linux world using distributions, like Ubuntu, that make it easy to complete many tasks using a graphical user interface (GUI), may speak with trepidation about the mysteries of the text interface. Others will either praise its power or comment about doing something via the command line as if it is the most natural and obvious way to complete a task.

It is not necessary for you to embrace either extreme. You may develop an affinity for the command line when performing some tasks and prefer the GUI for others. This is where most users end up today. Some may say that you will never need to access the command line because Ubuntu offers a slew of graphical tools that enable you to configure most things on your system. There are some good reasons to acquire a fundamental level of comfort with the command line that you should consider before embracing that view.

Sometimes things go wrong, and you may not have the luxury of a graphical interface to work with. In these situations, a fundamental understanding of the command line and its uses can be a real lifesaver. Also, some tasks end up being far easier and faster to accomplish from the command line.

NOTE

Don't be tempted to skip over this chapter as irrelevant. You should take the time to work through the chapter and ensure that you are comfortable with the command line before moving on. Doing so will benefit you greatly for years to come.

Initially, some may be tempted to think of the command line as the product of some sort of black and arcane art, and in some ways it can appear to be extremely difficult and complicated to use. However, with a little perseverance, by the end of this chapter you will start to feel comfortable using the command line and you'll be ready to move on to Chapter 35, "Command Line Masterclass."

More importantly, though, you will be able to make your way around a command line–based system, which you are likely to encounter if you ever work with a Linux server since many Linux servers have no GUI and all administration is done using a command line interface via SSH.

This chapter introduces you to commands that enable you to do the following:

- **Perform routine tasks—**Logging in and out, changing passwords, listing and navigating file directories
- **Implement basic file management—**Creating files and folders, copying or moving them around the file system, renaming and deleting them
- **Execute basic system management—**Shutting down or rebooting, using text-based tools to edit system configuration files, and reading man pages, which are entries for commands included as files already on your computer in a standardized manual format

The information in this chapter is valuable for individual users or system administrators who are new to Linux and are learning to use the command line for the first time.

TIP

Those of you who have used a computer for many years will probably have come into contact with MS-DOS, in which case being presented with a black screen will fill you with a sense of nostalgia. Don't get too comfy; the command line in Linux is different from (and actually more powerful than) its distant MS-DOS cousin. Even cooler is that, whereas MS-DOS skills are transferable only to other MS-DOS environments, the skills that you learn at the Linux command line can be transferred easily to other UNIX and UNIX-like operating systems, such as Solaris, OpenBSD, FreeBSD, and even Mac OS X, which provides access to the terminal.

User Accounts

A good place to start is with the concept of user-based security. For the most part, only two types of people will access the system as users (although there will be other accounts that run programs and processes, here we are talking about accounts that represent human beings rather than something like an account created for a web server process). Most people have a regular user account. These users can change anything that is specific to their account, such as the wallpaper on the desktop, their personal preferences, the configuration for a program when it is run by them using their account, and so on. Note that the emphasis is on anything that is specific to them. This type of user is not able to make systemwide changes that could affect other users.

To make systemwide changes, you need to use super-user privileges, such as the account you created when you started Ubuntu for the first time (see Chapter 1, "Installing Ubuntu"). With super-user privileges you have access to the entire system and can carry out any task, even destructive ones! To help prevent this from happening, this user does not run with these powers enabled at all times, but instead spends most of the time with the look and feel of a regular user.

To use your super-user privileges from the command line you need to preface the command you want to execute with another command, `sudo`, followed by a space and the command you want to run. As a mnemonic device, some think of this as "super-user do." When you press Enter (after typing the remaining command) you will be prompted for your password, which you should type in followed by the Enter key. As usual on any UNIX-based system, the password will not appear on the screen while you are typing it as a security measure, in case someone is watching over your shoulder. Ubuntu will then carry out the command, but with super-user privileges.

An example of the destructive nature of working as the super-user can be found in the age-old example, `sudo rm -rf /`, which erases everything on your hard drive. If you enter a command using `sudo` as a regular user who does not have an account with super-user privileges, an error message will appear and nothing will happen because the command will not run. We recommend that you don't try this particular command as a test, though. If you enter this command using an account with super-user privileges, you will soon find yourself starting over with a fresh installation and hoping you have a current backup of all of your data. You need to be especially careful when using your super-user privileges; otherwise, you may do irreparable damage to your system.

However, the ability to work as the super-user is fundamental to a healthy Linux system and should not be feared, but rather respected and used only with focused attention. Without this ability you would not be able to install new software, edit system configuration files, or do a large number of important administration tasks. By the end of this chapter you will feel comfortable working with your super-user privileges and be able to adequately administer your system from the command line. By the way, you have already been performing operations with super-user privileges from the GUI if you have ever been asked to enter your password to complete a specific task, such as installing software updates. The difference is that most graphical interfaces limit the options that users have and make it a little more difficult to do some of the big, disruptive tasks, even the ones that are incredibly useful.

Ubuntu works slightly differently from many other Linux distributions. If you study some other Linux distros, especially older or more traditional ones, you will hear about a specific user account called `root`, which is a super-user account. In those distros, instead of typing in `sudo` before a command while using a regular user account with super-user privileges, you log in to the `root` account and simply issue the command without entering a password. In those cases, you can tell when you are using the root account at the command line because you will see a pound sign (#) in the command line prompt in the place of the dollar sign ($).

For example, `matthew@seymour:~#` vs the usual `matthew@seymour:~$`.

In Ubuntu the `root` account is disabled by default because forcing regular users with super-user privileges to type a specific command every time they want to execute a command as a super-user should have the benefit of making them carefully consider what they are doing when they use that power. It is easy to forget to log out of a root account, and entering a powerful command while logged in to `root` can be catastrophic. However, if you are more experienced and comfortable with the more traditional method of using super-user privileges and want to enable the root account, you can use the command `sudo passwd`. When prompted, enter your user password to confirm that your user account has super-user privileges. You will then be asked for a new UNIX password, which will be the password for the root account, so make sure and remember it. You will also be prompted to repeat the password, in case you've made any mistakes. After you've typed it in and pressed Enter, the `root` account will be active. You'll find out how to switch to `root` later on.

An alternative way of getting a root prompt, without having to enable the root account, is to issue the command `sudo -i`. After entering your password you will find yourself at a root prompt (#). Do what you need to do and when you are finished, type `exit` and press Enter to return to your usual prompt. You can learn more about `sudo` and `root` from an Ubuntu perspective at https://help.ubuntu.com/community/RootSudo.

Ubuntu offers a number of ways to access the command line. One way is to press the key combination Ctrl+Alt+F1, after which Ubuntu will switch to a black screen and a login prompt like this:

```
Ubuntu 10.10 maverick seymour ttyl
seymour login:
```

TIP

This is one of six virtual consoles that Ubuntu provides. After you have accessed a virtual console, you can use the Ctrl+Alt + any of F1 through F6 to switch to a different console. If you want to get back to the graphical interface, press Ctrl+Alt+ F7. You can also switch between consoles by holding the Alt key and pressing either the left or the right cursor key to move down or up a console, such as tty1 to tty2.

At the login prompt enter your username and press the Enter key. You will be asked for your password, which you should enter. Note that Ubuntu does not show any characters while you are typing your password in. This is a good thing because it prevents any shoulder surfers from seeing what you've typed or the length of the password.

Pressing the Enter key drops you to a shell prompt, signified by the dollar sign:

```
matthew@seymour:~$
```

This particular prompt tells me that I am logged in as the user `matthew` on the system `seymour` and I am currently in my home directory; Linux uses the tilde (~) as shorthand for the home directory, which would usually be something like `/home/matthew`.

TIP

Navigating through the system at the command line can get confusing at times, especially when a directory name occurs in several places. Fortunately, Linux includes a simple command that tells you exactly where you are in the file system. It's easy to remember because the command is just an abbreviation of present working directory, so type **`pwd`** at any point to get the full path of your location. For example, typing **`pwd`** after following these instructions shows `/home/yourusername`, meaning that you are currently in your home directory.

Using the `pwd` command can save you a lot of frustration when you have changed directory half a dozen times and have lost track.

Another way to quickly access the terminal is to use the desktop menu option Applications, Accessories, Terminal. This opens up `gnome-terminal`, which allows you to access the terminal while remaining in a GUI environment. This time, the terminal appears as white text on an aubergine (dark purple) background. This is the most common method for most desktop users, but both are useful.

Regardless of which way you access the terminal, using the virtual tty consoles accessible at Ctrl + Alt + F1-6 or via the windowed version atop your GUI desktop, you will find the rest of the usage details that we cover work the same. As you continue to learn and experiment beyond the contents of this book, you may start to discover some subtle differences between the two and develop a preference. For our purposes, either method will work quite well.

With that, let's begin.

Navigating Through the File System

Use the `cd` command to navigate the file system. This command is generally used with a specific directory location or pathname, like this:

```
matthew@seymour:~$ cd /etc/apt/
```

The `cd` command can also be used with several shortcuts. For example, to quickly move up to the *parent* directory, the one above the one you are currently in, use the cd command like this:

```
matthew@seymour:~$ cd ..
```

To return to your home directory from anywhere in the Linux file system, use the cd command like this:

```
matthew@seymour:~$ cd
```

You can also use the `$HOME` shell environment variable to accomplish the same thing. Type this command and press Enter to return to your home directory:

```
matthew@seymour:~$ cd $HOME
```

You can accomplish the same thing by using the tilde (~) like this:

```
matthew@seymour:~$ cd ~
```

Don't forget you can always use `pwd` to remind you where you are within the file system!

The `ls` command lists the contents of the current directory. It's commonly used by itself, but a number of options (also known as switches) are available for `ls` and give you more information. For instance, the following command returns a listing of all the files and directories within the current directory, including any hidden files (denoted by a `.` prefix) as well as a full listing, so it will include details such as the permissions, owner and group, size, and last modified time and date:

```
matthew@seymour:~$ ls -la
```

You can also issue the command

```
matthew@seymour:~$ ls -R
```

This command scans and lists all the contents of the subdirectories of the current directory. This might be a lot of information, so you may want to redirect the output to a text file so you can browse through it at your leisure by using the following:

```
matthew@seymour:~$ ls -laR > listing.txt
```

TIP

The previous command sends the output of `ls -laR` to a file called `listing.txt` and demonstrates part of the power of the Linux command line. At the command line you are able to use files as inputs to commands, or generate files as outputs as shown. For more information about combining commands, see Chapter 30. In the meantime, note that you can read the contents of that text file using the command `less listing.txt`, which will let you read the file bit by bit using the arrow keys to navigate in the file (or `Enter` to move to the next line), the spacebar to move to the next page, and q to exit when done.

Table 4.1 shows some of the top-level directories that are part of a standard Linux distro.

Table 4.1. Basic Linux Directories

Name	Description
/	The root directory
/bin	Essential commands
/boot	Boot loader files, Linux kernel
/dev	Device files
/etc	System configuration files
/home	User home directories
/lib	Shared libraries, kernel modules
/lost+found	Directory for recovered files (if found after a file system check)
/media	Mount point for removable media, such as DVDs and floppy disks
/mnt	Usual mount point for local, remote file systems
/opt	Add-on software packages
/proc	Kernel information, process control
/root	Super-user (root) home
/sbin	System commands (mostly root only)
/srv	Holds information relating to services that run on your system
/sys	Real-time information on devices used by the kernel
/tmp	Temporary files
/usr	Software not essential for system operation, such as applications
/var	Variable data (such as logs); spooled files

Knowing these directories can help you find files when you need them. This knowledge can even help you partition hard drives when you install new systems by letting you choose to put certain directories on their own distinct partition, which can be useful for things like isolating directories from one another, such as a server security case like putting a directory like /boot that doesn't change often on its own partition and making it read-only and unchangeable without specific operations being done by a super-user during a maintenance cycle. Desktop users probably won't need to think about that, but the directory

tree is still quite useful to know when you want to find the configuration file for a specific program and set some program options systemwide to affect all users.

Some of the important directories in Table 4.1, such as those containing user and root commands or system configuration files, are discussed in the following sections. You may use and edit files under these directories when you use Ubuntu.

Linux also includes a number of commands you can use to search the file system. These include the following:

- `whereis` *`command`* —Returns the location of the command (e.g. /bin, /sbin, or /usr/bin/command) and its man page, which is an entry for the command included as a file already on your computer in a standardized manual format.
- `whatis` *`command`* —Returns a one-line synopsis from the command's man page.
- `locate` *`file`* —Returns locations of all matching file(s); an extremely fast method of searching your system because `locate` searches a database containing an index of all files on your system. However, this database (which is several MB in size and named `mlocate.db`, under the `/var/lib/mlocate` directory) is built daily at 6:25 a.m. by default, and does not contain pathnames to files created during the workday or in the evening. If you do not keep your machine on constantly, you can run the `updatedb` command with super-user privileges to manually start the building of the database. More advanced users can change the time the database is updated by changing a `cron` job that calls the command. There is a script in `/etc/cron.daily` called `mlocate` that does this as part of the daily maintenance for the database. You would need to either remove that script from `/etc/cron.daily` and put it the root `crontab` or change when the daily tasks are run. If you don't know how to do this right now, you should wait until you have read and understand Chapter 11, "Automating Tasks," and its discussion of `cron`, after which the preceding description will be clear.
- `apropos` *`subject`* —Returns a list of commands related to subject.
- `type` *`name`* —Returns how a name would be interpreted if used as a command. This generally shows options or the location of the binary that will be used. For example, type ls returns ls is aliased to `` `ls –color=auto' ``.

Managing Files with the Shell

Managing files in your home directory involves using one or more easily remembered commands. Basic file management operations include paging (reading), moving, renaming, copying, searching, and deleting files and directories. Here are some commands to do these tasks:

- `cat` *`filename`* —Outputs contents of filename to display
- `less` *`filename`* —Allows scrolling while reading contents of filename
- `mv` *`file1 file2`* —Renames *`file1`* to *`file2`* and may be used with full pathnames to move the file to a new directory at the same time
- `mv` *`file dir`* —Moves file to specified directory
- `cp` *`file1 file2`* —*Copies* `file1` and creates `file2`
- `rm` *`file`* —Deletes file
- `rmdir` *`dir`* —Deletes directory (if empty)
- `grep` *`string file`* `(s)`—Searches through files(s) and displays lines containing matching string (a string is a series of characters like letters, numbers, and symbols, such as a word)

Note that each of these commands can be used with pattern-matching strings known as *wildcards* or *expressions*. For example, to delete all files in the current directory beginning with the letters `abc`, you

can use an expression beginning with the first three letters of the desired filenames. An asterisk (`*`) is then appended to match all these files. Use a command line with the `rm` command like this:

```
matthew@seymour:~$ rm abc*
```

Linux shells recognize many types of filenaming wildcards, but this is different from the capabilities of Linux commands supporting the use of more complex expressions. You learn more about using wildcards in Chapter 11, "Automating Tasks."

NOTE

You can also learn more about using expressions by reading the grep manual pages (`man grep`).

Working with Compressed Files

Another file management operation is compression and decompression of files, or the creation, listing, and expansion of file and directory archives. Linux distributions usually include several compression utilities you can use to create, compress, expand, or list the contents of compressed files and archives. These commands include:

- *`bunzip2`*—Expands a compressed file
- *`bzip2`*—Compresses or expands files and directories
- *`gunzip`*—Expands a compressed file
- *`gzip`*—Compresses or expands files and directories
- *`tar`*—Creates, expands, or lists the contents of compressed or uncompressed file or directory archives known as *tape archives* or *tarballs*

Most of these commands are easy to use. However, the `tar` command, which is the most commonly used of the bunch, has a somewhat complex set of command-line options and syntax. This flexibility and power are part of its popularity; you can quickly learn to use `tar` by remembering a few of the simple command line options. For example, to create a compressed archive of a directory, use `tar`'s `czf` options like this:

```
matthew@seymour:~$ tar czf dirname.tgz dirname
```

The result is a compressed archive (a file ending in `.tgz`) of the specified directory (and all files and directories under it). Add the letter `v` to the preceding options to view the list of files added during compression and archiving while the archive is being created. To list the contents of the compressed archive, substitute the c option with the letter `t`, like this:

```
matthew@seymour:~$ tar tzf archive
```

However, if many files are in the archive, a better invocation (to easily read or scroll through the output) is

```
matthew@seymour:~$ tar tzf archive | less
```

TIP

In the previous code example, we used a pipe character (|). Each pipe sends the output of the first command to the next command. This is another of the benefits of the command line under Linux—you can string several commands together to get the desired results.

To expand the contents of a compressed archive, use `tar`'s `zxf` options, like so:

```
matthew@seymour:~$ tar zxf archive
```

The `tar` utility decompresses the specified archive and extracts the contents in the current directory.

Essential Commands from the `/bin` and `/sbin` Directories

The `/bin` directory contains essential commands used by the system for running and booting the system. In general, only the root operator uses the commands in the `/sbin` directory. The software in both locations is essential to the system; they make the system what it is, and if they are changed or removed, it could cause instability or a complete system failure. Often, the commands in these two directories are *statically* linked, which means that the commands do not depend on software libraries residing under the `/lib` or `/usr/lib` directories. Nearly all the other applications on your system are *dynamically* linked—meaning that they require the use of external software libraries (also known as *shared* libraries) to run. This is a feature for both sets of software.

The commands in `/bin` and `/sbin` are kept stable to maintain foundational system integrity and do not need to be updated often, if at all. For the security of the system, these commands are kept in a separate location and isolated where changes are more difficult and where it will be more obvious to the system administrator if unauthorized changes are attempted or made.

Application software changes more frequently, and applications often use the same functions that other pieces of application software use. This was the genesis of shared libraries. When a security update is needed for something that is used by more than one program, it has to be updated in only one location, a specific software library. This enables easy and quick security updates that will affect several pieces of non-system-essential software at the same time by updating one shared library, contained in one file on the computer.

Using and Editing Files in the `/etc` Directory

System configuration files and directories reside under the `/etc` directory. Some major software packages, such as Apache, OpenSSH, and `xinetd`, have their own subdirectories in `/etc` filled with configuration files. Others like crontab or fstab use one file. Some examples of system-related configuration files in `/etc` include the following:

- *`fstab`*—The file system table is a text file listing each hard drive, CD-ROM, floppy, or other storage device attached to your PC. The table indexes each device's partition information with a place in your Linux file system (directory layout) and lists other options for each device when used with Linux (see Chapter 36, "Kernel and Module Management"). Nearly all entries in `fstab` can be manipulated by root using the `mount` command.
- *`modprobe.d/`*—This folder holds all the instructions to load kernel modules that are required as part of the system startup.
- *`passwd`*—The list of users for the system, including special-purpose nonhuman users like syslog and couchdb, along with user account information.
- *`sudoers`*—A list of users or user groups with super-user access.

Protecting the Contents of User Directories—`/home`

The most important data on a Linux system resides in the user's directories, found under the `/home` directory. Segregating the system and user data can be helpful in preventing data loss and making the process of backing up easier. For example, having user data reside on a separate file system or mounted from a remote computer on the network might help shield users from data loss in the event of a system hardware failure. For a laptop or desktop computer at home, you might place `/home` on a separate partition from the rest of the file system, so that if the operating system is upgraded, damaged, or reinstalled, `/home` would be more likely to survive the event intact.

Using the Contents of the `/proc` Directory to Interact with the Kernel

The content of the `/proc` directory is created from memory and exists only while Linux is running. This directory contains special files that either extract information from or send information to the kernel. Many Linux utilities extract information from dynamically created directories and files under this directory, also known as a *virtual file system*. For example, the `free` command obtains its information from a file named **`meminfo:`**

```
matthew@seymour:~$ free

             total       used       free     shared    buffers     cached
Mem:       4055680    2725684    1329996          0     188996    1551464
-/+ buffers/cache:     985224    3070456
Swap:      8787512          0    8787512
```

This information constantly changes as the system is used. You can get the same information by using the cat command to see the contents of the `meminfo` file:

```
matthew@seymour:~$ cat /proc/meminfo
MemTotal:                4055680 KB
MemFree:                 1329692 KB
Buffers:                  189208 KB
Cached:                  1551488 KB
SwapCached:                    0 KB
Active:                  1222172 KB
Inactive:                1192244 KB
Active(anon):             684092 KB
Inactive(anon):               16 KB
Active(file):             538080 KB
Inactive(file):          1192228 KB
Unevictable:                  48 KB
Mlocked:                      48 KB
SwapTotal:               8787512 KB
SwapFree:                8787512 KB
Dirty:                       136 KB
Writeback:                     0 KB
AnonPages:                673760 KB
Mapped:                   202308 KB
Shmem:                     10396 KB
Slab:                     129248 KB
SReclaimable:             107356 KB
SUnreclaim:                21892 KB
KernelStack:                2592 KB
PageTables:                30108 KB
NFS_Unstable:                  0 KB
Bounce:                        0 KB
WritebackTmp:                  0 KB
CommitLimit:             10815352 KB
```

```
Committed_AS:               1553172 KB
VmallocTotal:           34359738367 KB
VmallocUsed:                342300 KB
VmallocChunk:           34359387644 KB
HardwareCorrupted:              0 KB
HugePages_Total:                   0
HugePages_Free:                    0
HugePages_Rsvd:                    0
HugePages_Surp:                    0
Hugepagesize:                   2048 KB
DirectMap4k:                   38912 KB
DirectMap2M:                 4153344 KB
```

The `/proc` directory can also be used to dynamically alter the behavior of a running Linux kernel by "echoing" numerical values to specific files under the `/proc/sys` directory. For example, to "turn on" kernel protection against one type of denial of service (DOS) attack known as *SYN flooding*, use the echo command to send the number 1 (one) to the following `/proc` path:

```
matthew@seymour:~$ $ sudo echo 1 >/proc/sys/net/ipv4/tcp_syncookies
```

Other ways to use the /proc directory include the following:

- Getting CPU information, such as the family, type, and speed from `/proc/cpuinfo`.
- Viewing important networking information under `/proc/net`, such as active interfaces information under `/proc/net/dev`, routing information in `/proc/net/route`, and network statistics in `/proc/net/netstat`.
- Retrieving file system information.
- Reporting media mount point information via USB; for example, the Linux kernel reports what device to use to access files (such as `/dev/sda`) if a USB camera or hard drive is detected on the system. You can use the `dmesg` command to see this information.
- Getting the kernel version in `/proc/version`, performance information such as uptime in `/proc/uptime`, or other statistics such as CPU load, swap file usage, and processes in `/proc/stat`.

Working with Shared Data in the `/usr` Directory

The `/usr` directory contains software applications, libraries, and other types of shared data for use by anyone on the system. Many Linux system administrators give `/usr` its own partition. A number of subdirectories under `/usr` contain manual pages (`/usr/share/man`), software package shared files (`/usr/share/` *`name_of_package`* , *such as `/usr/share/emacs`*), additional application or software package documentation (`/usr/share/doc`), and an entire subdirectory tree of locally built and installed software, `/usr/local`.

Temporary File Storage in the `/tmp` Directory

As its name implies, the `/tmp` directory is used for temporary file storage; as you use Linux, various programs create files in this directory.

Accessing Variable Data Files in the `/var` Directory

The `/var` directory contains subdirectories used by various system services for spooling and logging. Many of these variable data files, such as print spooler queues, are temporary, whereas others, such as

system and kernel logs, are renamed and rotated in use. Incoming electronic mail is usually directed to files under `/var/spool/mail`.

Linux also uses `/var` for other important system services. These include the top-most File Transfer Protocol (FTP) directory under `/var/ftp` (see Chapter 18, "Remote File Serving with FTP"), and the Apache web server's initial home page directory for the system, `/var/www/html`. (See Chapter 17, "Apache Web Server Management," for more information on using Apache.)

NOTE

There is a recent trend to move data that is served from /var/www and /var/ftp to /srv, but this is not universal.

Logging In to and Working with Linux

You can access and use a Linux system in a number of ways. One way is at the console with a monitor, keyboard, and mouse attached to the PC. Another way is via a serial console, either by dial-up via a modem or a PC running a terminal emulator and connected to the Linux PC via a null modem cable. You can also connect to your system through a wired or wireless network, using the `telnet` or `ssh` commands. The information in this section shows you how to access and use the Linux system, using physical and remote text-based logins.

NOTE

This chapter focuses on text-based logins and use of Linux. Graphical logins and using a graphical desktop are described in Chapter 3, "Working with GNOME."

Text-Based Console Login

If you sit down at your PC and log in to a Linux system that has not been booted to a graphical login, you see a prompt similar to this one:

```
Ubuntu 10.10 maverick seymour ttyl
seymour login:
```

Your prompt might vary, depending on the version of Ubuntu you are using and the method you are using to connect. In any event, at this prompt, type in your username and press Enter. When you are prompted for your password, type it in and press Enter.

NOTE

Your password is not echoed back to you, which is a good idea. Why is it a good idea? Well, people are prevented from looking over your shoulder and seeing your screen input. It is not difficult to guess that a five-letter password might correspond to the user's spouse's first name!

Logging Out

Use the `exit` or `logout` command or Ctrl+D to exit your session. You are then returned to the login prompt. If you use virtual consoles, remember to exit each console before leaving your PC. (Otherwise, someone could easily sit down and use your account.)

Logging in and Out from a Remote Computer

Although you can happily log in on your computer, an act known as a *local* login, you can also log in to your computer via a network connection from a remote computer. Linux-based operating systems provide a number of remote access commands you can use to log in to other computers on your local area network (LAN), wide area network (WAN), or the Internet. Note that not only must you have an account on the remote computer, but the remote computer must be configured to support remote logins—otherwise, you won't be able to log in.

NOTE

See Chapter 14, "Networking," to see how to set up network interfaces with Linux to support remote network logins and Chapter 11 to see how to start remote access services (such as `sshd`).

The best and most secure way to log in to a remote Linux computer is to use `ssh`, the Secure Shell client. Your login and session are encrypted while you work on the remote computer. The `ssh` client features many command-line options, but can be simply used with the name or IP address of the remote computer, like this:

```
matthew@seymour:~$ ssh 192.168.0.41
The authenticity of host '192.168.0.41 (192.168.0.41)' can't be established.
RSA key fingerprint is e1:db:6c:da:3f:fc:56:1b:52:f9:94:e0:d1:1d:31:50.
Are you sure you want to continue connecting (yes/no)?
yes
```

The first time you connect with a remote computer using `ssh`, Linux displays the remote computer's encrypted identity key and asks you to verify the connection. After you type `yes` and press Enter, you are warned that the remote computer's identity (key) has been entered in a file named `known_hosts` under the `.ssh` directory in your home directory. You are also prompted to enter your password:

```
Warning: Permanently added '192.168.0.41' (RSA) to the list of known hosts.
matthew@192.168.0.41's password:
matthew@babbage~$
```

After entering your password, you can work on the remote computer, which you can confirm by noticing the changed prompt that now uses the name of the remote computer on which you are working. Again, because you are using `ssh`, everything you enter on the keyboard in communication with the remote computer is encrypted. When you log out, you will return to the shell on your computer.

```
matthew@babbage~$ logout
matthew@seymour:~$
```

Using Environment Variables

A number of in-memory variables are assigned and loaded by default when you log in. These variables are known as shell *environment variables and* can be used by various commands to get information about your environment, such as the type of system you are running, your home directory, and the shell in use. Environment variables are used to help tailor the computing environment of your system and include helpful specifications and setup, such as default locations of executable files and software libraries. If you begin writing shell scripts, you might use environment variables in your scripts. Until then, you need to be aware only of what environment variables are and do.

The following list includes a number of environment variables, along with descriptions of how the shell uses them:

- `PWD`—provides the full path to your current working directory, used by the `pwd` command, such as `/home/matthew/Documents`
- `USER`—declares the user's name, such as `matthew`
- `LANG`—sets the default language, such as English
- `SHELL`—declares the name and location of the current shell, such as `/bin/bash`
- `PATH`—sets the default location(s) of executable files, such as `/bin`, `/usr/bin`, and so on
- `TERM`—sets the type of terminal in use, such as `vt100`, which can be important when using screen-oriented programs, such as text editors
- You can print the current value of any environment variable using `echo $VARIABLENAME`, like this:

```
matthew@seymour:~$ echo $USER

matthew
matthew@seymour:~$
```

NOTE

Each shell can have its own feature set and language syntax, as well as a unique set of default environment variables. See Chapter 15, "Remote Access for SSH and Telnet," for more information about using the different shells included with Ubuntu.

You can use the `env` or `printenv` command to display all environment variables, like so:

```
matthew@seymour:~$ env
ORBIT_SOCKETDIR=/tmp/orbit-matthew
SSH_AGENT_PID=1729
TERM=xterm
SHELL=/bin/bash
WINDOWID=71303173
GNOME_KEYRING_CONTROL=/tmp/keyring-qTEFTw
GTK_MODULES=canberra-gtk-module
USER=matt
hew
SSH_AUTH_SOCK=/tmp/keyring-qTEFTw/ssh
DEFAULTS_PATH=/usr/share/gconf/gnome.default.path
SESSION_MANAGER=local/seymour:/tmp/.ICE-unix/1695
USERNAME=matthew
XDG_CONFIG_DIRS=/etc/xdg/xdg-gnome:/etc/xdg
DESKTOP_SESSION=gnome
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
PWD=/home/matt
hew
GDM_KEYBOARD_LAYOUT=us
LANG=en_US.utf8
GNOME_KEYRING_PID=1677
MANDATORY_PATH=/usr/share/gconf/gnome.mandatory.path
GDM_LANG=en_US.utf8
GDMSESSION=gnome
HISTCONTROL=ignoreboth
SPEECHD_PORT=7560
SHLVL=1
HOME=/home/matt
hew
LOGNAME=matt
hew
```

```
LESSOPEN=| /usr/bin/lesspipe %s
DISPLAY=:0.0
LESSCLOSE=/usr/bin/lesspipe %s %s
XAUTHORITY=/var/run/gdm/auth-for-matthew-PzcGqF/database
COLORTERM=gnome-terminal
OLDPWD=/var/lib/mlocate
_=/usr/bin/env
```

This abbreviated list shows some of the common variables. These variables are set by configuration or *resource* files contained in `the /etc, /etc/skel`, or `user /home directory`. You can find default settings for bash, for example, `in /etc/profile` and `/etc/bashrc` as well as `.bashrc` or `.bash_profile` files in your home directory. Read the man page for `bash` for details about using these configuration files.

One of the most important environment variables is `$PATH`, which defines the location of executable files. For example, if, as a regular user, you try to use a command that is not located in your `$PATH` (such as the imaginary `command` command), you will see something like this:

```
matthew@seymour:~$ command
-bash: command: command not found
```

If the command that you're trying to execute exists in the Ubuntu software repositories, but is not yet installed on your system, Ubuntu will respond with the correct command to install the command.

```
matthew@seymour:~$ command

The program 'command' is currently not installed. You can install it by typing:

sudo apt-get install command
```

However, you might know that `command` is definitely installed on your system, and you can verify this by using the `whereis` command, like so:

```
$ whereis command
command: /sbin/command
```

You can also run the command by typing its full pathname or complete directory specification, like this:

```
$ /sbin/command
```

As you can see in this example, the command `command` is indeed installed. What happened is that by default, the `/sbin` directory is not in your $PATH. One of the reasons for this is that commands under the `/sbin` directory are normally intended to be run only by root. You can add /sbin to your $PATH by editing the file `.bash_profile` in your home directory (if you use the bash shell by default, like most Linux users). Look for the following line:

```
PATH=$PATH:$HOME/bin
```

You can then edit this file, perhaps using the `vi` editor (discussed later in this chapter), to add the `/sbin` directory like so:

```
PATH=$PATH:/sbin$HOME/bin
```

Save the file. The next time you log in, the /sbin directory is in your $PATH. One way to use this change right away is to read in the new settings in .bash_profile by using the bash shell's source command like so:

```
$ source .bash_profile
```

You can now run commands located in the /sbin directory without the need to explicitly type the full pathname.

Some Linux commands also use environment variables, for example, to acquire configuration information (such as a communications program looking for a variable such as BAUD_RATE, which might denote a default modem speed).

To experiment with the environment variables, you can modify the PS1 variable to manipulate the appearance of your shell prompt. If you are working with bash, you can use its built-in export command to change the shell prompt. For example, if your default shell prompt looks like this:

```
matthew@seymour:~$
```

You can change its appearance by using the PS1 variable like this:

```
$ PS1='$OSTYPE r00lz ->'
```

After you press Enter, you see

```
linux-gnu r00lz ->
```

Notes

See the bash man page for other variables you can use for prompt settings.

Using the Text Editors

Linux distributions include a number of applications known as *text editors* that you can use to create text files or edit system configuration files. Text editors are similar to word processing programs, but generally have fewer features, work only with text files, and might or might not support spell checking or formatting. Text editors range in features and ease of use and are found on nearly every Linux distribution. The number of editors installed on your system depends on what software packages you've installed on the system.

Some of the more popular console-based text editors include:

- *emacs*—The comprehensive GNU emacs editing environment, which is much more than an editor; see the section "Working with emacs" later in this chapter.
- *joe*—Joe's Own Editor, a text editor, which can be used to emulate other editors.
- *nano*—A simple text editor similar to the pico text editor included with the pine email program.
- *vim*—An improved, compatible version of the vi text editor (which we call vi in the rest of this chapter because it has a symbolic link named vi and a symbolically linked manual page).

Note that not all text editors are *screen oriented*, meaning designed for use from a terminal. Some of the text editors are designed to run from a graphical desktop and which provide a graphical interface with menu bars, buttons, scrollbars, and so on, are:

- *gedit*—A GUI text editor for GNOME
- *kate*—A simple KDE text editor
- *kedit*—Another simple KDE text editor

A good reason to learn how to use a text-based editor, such as `vi or nano`, is that system maintenance and recovery operations almost never take place during X Window sessions, negating the use of a GUI editor. Many larger, more complex and capable editors do not work when Linux is booted to its single-user or maintenance mode. If anything does go wrong with your system and you can't log into the X Window system and its graphical user interface, knowledge and experience of using both the command line and text editors will turn out to be very important. Make a point of opening some of the editors and playing around with them; you never know—you might just thank me someday!

Another reason to learn how to use a text-based editor under the Linux console mode is so that you can edit text files through remote shell sessions, because many servers will not host graphical desktops.

Working with `vi`

The one editor found on nearly every UNIX and Linux system is the `vi` editor, originally written by Bill Joy. This simple-to-use but incredibly capable editor features a somewhat cryptic command set, but you can put it to use with only a few commands. Although many experienced UNIX and Linux users use `vi` extensively during computing sessions, many users who do only quick and simple editing might not need all its power and may prefer an easier-to-use text editor such as `pico` or GNU `nano`. Diehard GNU fans and programmers definitely use `emacs`.

However, learning how to use `vi` is a good idea. You might need to edit files on a Linux system with a minimal install, or a remote server without a more extensive offering of installed text editors. Chances are nearly 100 percent that `vi` will be available.

You can start an editing session by using the `vi` command like this:

```
matthew@seymour:~$ vi file.txt
```

The vi command works by using an insert (or editing) mode, and a viewing (or command) mode.

When you first start editing, you are in the viewing mode. You can use your arrow or other navigation keys (as shown later) to scroll through the text. To start editing, press the `i` key to insert text or the `a` key to append text. When you're finished, use the Esc key to toggle out of the insert or append modes and into the viewing (or command) mode. To enter a command, type a colon (`:`), followed by the command, such as `w` to write the file, and press Enter.

Although `vi` supports many complex editing operations and numerous commands, you can accomplish work by using a few basic commands. These basic `vi` commands are

- **Cursor movement**—`h, j, k, l` (left, down, up, and right)
- **Delete character**—`x`
- **Delete line**—`dd`
- **Mode toggle**—`Esc`, `Insert` (or `i`)
- **Quit**—`:q`
- **Quit without saving**—`:q!`
- **Run a shell command**—`:sh` (use `'exit'` to return)
- **Save file**—`:w`

- **Text search**—`/`

NOTE

Use the `vimtutor` command to quickly learn how to use `vi`'s keyboard commands. The tutorial takes less than 30 minutes, and it teaches new users how to start or stop the editor, navigate files, insert and delete text, and perform search, replace, and insert operations.

Working with `emacs`

Richard M. Stallman's GNU `emacs` editor, like `vi`, is included with Ubuntu and nearly every other Linux distribution. Unlike other UNIX and Linux text editors, `emacs` is much more than a simple text editor—it is an editing environment and can be used to compile and build programs and act as an electronic diary, appointment book, and calendar; use it to compose and send electronic mail, read Usenet news, and even play games. The reason for this capability is that `emacs` contains a built-in language interpreter that uses the Elisp (`emacs` LISP) programming language. `emacs` is not installed in Ubuntu by default. To use emacs, the package you need to install is called `emacs`. See Chapter 32, "Managing Software."

You can start an `emacs` editing session like this:

```
matthew@seymour:~$ emacs file.txt
```

TIP

If you start `emacs` when using X11, the editor launches in its own floating window. To force `emacs` to display inside a terminal window instead of its own window (which can be useful if the window is a login at a remote computer), use the `-nw` command-line option like this: `emacs -nw file.txt`.

The `emacs` editor uses an extensive set of keystroke and named commands, but you can work with it by using a basic command subset. Many of these basic commands require you to hold down the Ctrl key, or to first press a *meta* key (generally mapped to the Alt key). The basic commands are listed in Table 4.2.

Table 4.2. Emacs Editing Commands

Action	Command
Abort	Ctrl+G
Cursor left	Ctrl+B
Cursor down	Ctrl+N
Cursor right	Ctrl+F
Cursor up	Ctrl+P
Delete character	Ctrl+D
Delete line	Ctrl+K
Go to start of line	Ctrl+A
Go to end of line	Ctrl+E
Help	Ctrl+H
Quit	Ctrl+X, Ctrl+C
Save As	Ctrl+X, Ctrl+W
Save file	Ctrl+X, Ctrl+S
Search backward	Ctrl+R

Action	Command
Search forward	Ctrl+S
Start tutorial	Ctrl+H, T
Undo	Ctrl+X, U

TIP

One of the best reasons to learn how to use `emacs` is that you can use nearly all the same keystrokes to edit commands on the `bash` shell command line. Another reason is that like `vi`, `emacs` is universally available for installation on nearly every UNIX and Linux system, including Apple's Mac OS X.

Working with Permissions

Under Linux (and UNIX), everything in the file system, including directories and devices, is a file. And every file on your system has an accompanying set of permissions based on ownership. These permissions provide data security by giving specific permission settings to every single item denoting who may read, write, and/or execute the file. These permissions are set individually for the file's owner, for members of the group the file belongs to, and for all others on the system.

You can examine the default permissions for a file you create by using the `umask` command, which lists default permissions using the number system explained next, or by using the `touch` command and then the `ls` command's long-format listing like this:

```
matthew@seymour:~$ touch file
matthew@seymour:~$ ls -l file
-rw-r--r-- 1 matthew matthew 0 2010-06-30 13:06 file
```

In this example, the touch command is used to quickly create a file. The ls command then reports on the file, displaying the following (from left to right):

- **The type of file created—**Common indicators of the type of file are in the leading letter in the output. A blank (which is represented by a dash, as in the preceding example) designates a plain file, `d` designates a directory, `c` designates a character device (such as `/dev/ttyS0`), and `b` is used for a block device (such as `/dev/sda`).
- **Permissions—**Read, write, and execute permissions for the owner, group, and all others on the system. (You learn more about these permissions later in this section.)
- **Number of links to the file—**The number one (`1`) designates that there is only one file, whereas any other number indicates that there might be one or more hard-linked files. Links are created with the `ln` command. A hard-linked file is an exact copy of the file, but it might be located elsewhere on the system. Symbolic links of directories can also be created, but only the root operator can create a hard link of a directory.
- **The owner—**The account that owns the file; this is originally the file creator, but you can change this designation using the `chown` command.
- **The group—**The group of users allowed to access the file; this is originally the file creator's main group, but you can change this designation using the `chgrp` command.
- **File size and creation/modification date—**The last two elements indicate the size of the file in bytes and the date the file was created or last modified.

Assigning Permissions

Under Linux, permissions are grouped by owner, group, and others, with read, write, and execute permission assigned to each, like so:

```
Owner    Group    Others
Rwx      rwx      rxw
```

Permissions can be indicated by mnemonic or octal characters. Mnemonic characters are

- `r` indicates permission for an owner, member of the owner's group, or others to open and read the file.
- `w` indicates permission for an owner, member of the owner's group, or others to open and write to the file.
- `x` indicates permission for an owner, member of the owner's group, or others to execute the file (or read a directory).

In the previous example for the file named `file`, the owner, `matthew`, has read and write permission. Any member of the group named `matthew` may only read the file. All other users may only read the file. Also note that default permissions for files created by the root operator (while using sudo or a root account) will be different because of `umask` settings assigned by the shell.

Many users prefer to use numeric codes, based on octal (base 8) values, to represent permissions. Here's what these values mean:

- `4` indicates read permission.
- `2` indicates write permission.
- `1` indicates execute permission.

In octal notation, the previous example file has a permission setting of `664` (read + write or `4` + `2`, read + write or `4` + `2`, read-only or `4`). Although you can use either form of permissions notation, octal is easy to use quickly after you visualize and understand how permissions are numbered.

NOTE

In Linux, you can create groups to assign a number of users access to common directories and files, based on permissions. You might assign everyone in accounting to a group named `accounting` and allow that group access to accounts payable files while disallowing access by other departments. Defined groups are maintained by the root operator, but you can use the `newgrp` command to temporarily join other groups to access files (as long as the root operator has added you to the other groups). You can also allow or deny other groups' access to your files by modifying the group permissions of your files.

Directory Permissions

Directories are also files under Linux. For example, again use the `ls` command to show permissions like this:

```
matthew@seymour:~$ mkdir directory
matthew@seymour:~$ ls -ld directory
drwxr-xr-x                      2 matthew  matthew  4096 2010-06-30 13:23 directory
```

In this example, the `mkdir` command is used to create a directory. The `ls` command and its `-ld` option is used to show the permissions and other information about the directory (not its contents). Here you can

see that the directory has permission values of `755` (read + write + execute or `4 + 2 + 1`, read + execute or `4 + 1`, and read + execute or `4 + 1`).

This shows that the owner can read and write to the directory and, because of execute permission, also list the directory's contents. Group members and all other users can list only the directory contents. Note that directories require execute permission for anyone to be able to view their contents.

You should also notice that the `ls` command's output shows a leading `d` in the permissions field. This letter specifies that this file is a directory; normal files have a blank field in its place. Other files, such as those specifying a block or character device, have a different letter.

For example, if you examine the device file for a Linux serial port, you will see:

```
matthew@seymour:~$ ls -l /dev/ttyS0
crw-rw---- 1 root dialout 4, 64 2010-06-30 08:13 /dev/ttyS0
```

Here, `/dev/ttyS0` is a character device (such as a serial communications port and designated by a `c`) owned by root and available to anyone in the `dialout` group. The device has permissions of `660` (read + write, read + write, no permission).

On the other hand, if you examine the device file for an IDE hard drive, you see:

```
matthew@seymour:~$ ls -l /dev/sda
brw-rw---- 1 root disk 8, 0 2010-06-30 08:13 /dev/sda
```

In this example, `b` designates a block device (a device that transfers and caches data in blocks) with similar permissions. Other device entries you will run across on your Linux system include symbolic links, designated by `s`.

You can use the `chmod` command to alter a file's permissions. This command uses various forms of command syntax, including octal or a mnemonic form (such as `u`, `g`, `o`, or `a` and `rwx`, and so on) to specify a desired change. The `chmod` command can be used to add, remove, or modify file or directory permissions to protect, hide, or open up access to a file by other users (except for the root account or a user with super-user permission and using sudo, either of which can access any file or directory on a Linux system).

The mnemonic forms of `chmod`'s options are (when used with a plus character, `+`, to add, or a minus sign, `-`, to remove):

- *u*—Adds or removes user (owner) read, write, or execute permission
- *g*—Adds or removes group read, write, or execute permission
- *o*—Adds or removes read, write, or execute permission for others not in a file's group
- *a*—Adds or removes read, write, or execute permission for all users
- *r*—Adds or removes read permission
- *w*—Adds or removes write permission
- *x*—Adds or removes execution permission

For example, if you create a file, such as a `readme.txt`, the file will have default permissions (set by the `umask` setting in `/etc/bashrc`) of

```
-rw-r--r-- 1 matthew matthew 0 2010-06-30 13:33 readme.txt
```

As you can see, you can read and write the file. Anyone else can only read the file (and only if it is outside your home directory, which will have read, write, and execute permission set only for you, the owner). You can remove all write permission for anyone by using `chmod`, the minus sign, and `aw` like so:

```
matthew@seymour:~$ chmod a-w readme.txt
matthew@seymour:~$ ls -l readme.txt
-r--r--r-- 1 matthew matthew 0 2010-06-30 13:33 readme.txt
```

Now, no one can write to the file (except you, if the file is in your home or `/tmp` directory because of directory permissions). To restore read and write permission for only you as the owner, use the plus sign and the `u` and `rw` options like so:

```
matthew@seymour:~$ chmod u+rw readme.txt
matthew@seymour:~$ ls -l readme.txt
-rw-r--r-- 1 matthew matthew 0 2010-06-30 13:33 readme.txt
```

You can also use the octal form of the `chmod` command, for example, to modify a file's permissions so that only you, the owner, can read and write a file. Use the `chmod` command and a file permission of `600`, like this:

```
matthew@seymour:~$ chmod 600 readme.txt
matthew@seymour:~$ ls -l readme.txt
-rw------- 1 matthew matthew 0 2010-06-30 13:33 readme.txt
```

If you take away execution permission for a directory, files might be hidden inside and may not be listed or accessed by anyone else (except the root operator, of course, who has access to any file on your system). By using various combinations of permission settings, you can quickly and easily set up a more secure environment, even as a normal user in your home directory.

Understanding Set User ID and Set Group ID Permissions

Two more types of permission are "set user ID", known as *suid*, and "set group ID," or *sgid*, permissions. These settings, when used in a program, enable any user running that program to have program owner or group owner permissions for that program. These settings enable the program to be run effectively by anyone, without requiring that each user's permissions be altered to include specific permissions for that program.

One commonly used program with suid permissions is the `passwd` command:

```
matthew@seymour:~$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 42856 2010-01-26 10:09 /usr/bin/passwd
```

This setting allows normal users to execute the command (as root) to make changes to a root-only accessible file, `/etc/passwd`.

You also can assign similar permission with the `chfn` command. This command allows users to update or change `finger` information in `/etc/passwd`. You accomplish this permission modification by using a leading `4` (or the mnemonic `s`) in front of the three octal values.

NOTE

Other files that might have suid or guid permissions include `at`, `rcp`, `rlogin`, `rsh`, `chage`, `chsh`, `ssh`, `crontab`, `sudo`, `sendmail`, `ping`, `mount`, and several UNIX-to-UNIX Copy (UUCP) utilities. Many programs (such as games) might also have this type of permission to access a sound device.

Files or programs that have suid or guid permissions can sometimes present security holes because they bypass normal permissions. This problem is compounded if the permission extends to an executable binary (a command) with an inherent security flaw because it could lead to any system user or intruder gaining root access. In past exploits, this typically happened when a user fed a vulnerable command with unexpected input (such as a long pathname or option); the command would fail, and the user would be presented a root prompt. Although Linux developers are constantly on the lookout for poor programming practices, new exploits are found all the time, and can crop up unexpectedly, especially in newer software packages that haven't had the benefit of peer developer review.

Savvy Linux system administrators keep the number of suid or guid files present on a system to a minimum. The `find` command can be used to display all such files on your system:

```
matthew@seymour:~$ sudo find / -type f -perm /6000 -exec ls -l {} \;
```

NOTE

The `find` command is quite helpful and can be used for many purposes, such as before or during backup operations. See the section "Using Backup Software" in Chapter 13, "Backing Up."

Note that the programs do not necessarily have to be removed from your system. If your users really do not need to use the program, you can remove the programs execute permission for anyone. You have to decide, as the root operator, whether your users are allowed, for example, to mount and unmount CD-ROMs or other media on your system. Although Linux-based operating systems can be set up to accommodate ease of use and convenience, allowing programs such as `mount` to be suid might not be the best security policy. Other candidates for suid permission change could include the `chsh`, `at`, or chage commands.

Working as Root

The root, or super-user account, is a special account and user on UNIX and Linux systems. Super-user permissions are required in part because of the restrictive file permissions assigned to important system configuration files. You must have root permission to edit these files or to access or modify certain devices (such as hard drives). When logged in as root, you have total control over your system, which can be dangerous.

When you work in root, you can destroy a running system with a simple invocation of the `rm` command like this:

```
matthew@seymour:~$ sudo rm -rf /
```

This command line not only deletes files and directories, but also could wipe out file systems on other partitions and even remote computers. This alone is reason enough to take precautions when using root access.

The only time you should run Linux as the super-user is when you are configuring the file system, for example, or to repair or maintain the system. Logging in and using Linux as the root operator isn't a good idea because it defeats the entire concept of file permissions.

Knowing how to run commands as the super-user (root) without logging in as root can help avoid serious missteps when configuring your system. In Ubuntu, you can use `sudo` to allow you to execute single commands as root and then quickly return to normal user status. For example, if you would like to edit

your system's file system table (a simple text file that describes local or remote storage devices, their type, and location), you can use `sudo` like this:

```
matthew@seymour:~$ sudo nano -w /etc/fstab
Password:
```

After you press Enter, you are prompted for a password that gives you access to root. This extra step can also help you "think before you leap" into the command. Enter the root password, and you are then editing `/etc/fstab`, using the nano editor with line wrapping disabled (thanks to the -w).

CAUTION

Before editing any important system or software service configuration file, make a backup copy. Then make sure to launch your text editor with line wrapping disabled. If you edit a configuration file without disabling line wrapping, you could insert spurious carriage returns and line feeds into its contents, causing the configured service to fail when restarting. By convention, nearly all configuration files are formatted for 80-character text width, but this is not always the case. By default, the `vi` and `emacs` editors don't use line wrap.

Creating Users

When a Linux system administrator creates a user, an entry in `/etc/passwd` for the user is created. The system also creates a directory, labeled with the user's username, in the `/home` directory. For example, if you create a user named `heather`, the user's home directory is `/home/heather`.

NOTE

In this chapter, you learn how to manage users from the command line. See Chapter 10, "Managing Users," for more information on user administration with Ubuntu using graphical administration utilities, such as the graphical `users-admin` client found in the menu at System, Administration, Users and Groups.

Use the `useradd` command, along with a user's name, to quickly create a user:

```
matthew@seymour:~$ sudo useradd ryan
```

After creating the user, you must also create the user's initial password with the `passwd` command:

```
matthew@seymour:~$ sudo passwd ryan


Changing password for user ryan.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

Enter the new password twice. If you do not create an initial password for a new user, the user cannot log in.

You can view `useradd`'s default new user settings by using the command and its `-D` option, like this:

```
matthew@seymour:~$ useradd -D
GROUP=100
HOME=/home
```

```
INACTIVE=-1
EXPIRE=
SHELL=/bin/sh
SKEL=/etc/skel
CREATE_MAIL_SPOOL=no
```

These options display the default group ID, home directory, account and password policy (active forever with no password expiration), the default shell, and the directory containing defaults for the shell.

The `useradd` command has many command-line options. The command can be used to set policies and dates for the new user's password, assign a login shell, assign group membership, and other aspects of a user's account. See `man useradd` for more info.

Deleting Users

Use the `userdel` command to delete users from your system. This command removes a user's entry in the system's `/etc/passwd` file. You should also use the command's `-r` option to remove all the user's files and directories (such as the user's mail spool file under `/var/spool/mail`):

```
matthew@seymour:~$ sudo userdel -r andrew
```

If you do not use the -r option, you have to manually delete the user's directory under `/home`, along with the user's `/var/spool/mail` queue.

Shutting Down the System

Use the `shutdown` command to shut down your system. The `shutdown` command has a number of different command-line options (such as shutting down at a predetermined time), but the fastest way to cleanly shut down Linux is to use the `-h` or halt option, followed by the word `now` or the numeral zero (`0`), like this:

```
matthew@seymour:~$ sudo shutdown -h now
```

or

```
matthew@seymour:~$ sudo shutdown -h 0
```

To incorporate a timed shutdown and a pertinent message to all active users, use shutdown's time and message options, like so:

```
matthew@seymour:~$ sudo shutdown -h 18:30 "System is going down for maintenance this
evening at 6:30 p.m. Please make sure you have saved your work and logged out by
then or you may lose data."
```

This example shuts down your system and provides a warning to all active users 15 minutes before the shutdown (or reboot). Shutting down a running server can be considered drastic, especially if there are active users or exchanges of important data occurring (such as a backup in progress). One good approach is to warn users ahead of time. This can be done by editing the system Message of the Day (MOTD) `motd` file, which displays a message to users when they login using the command line interface, as is common on multi-user systems.

It used to be that to create a custom MOTD you only had to use a text editor and change the contents of `/etc/motd`. However, this has changed in Ubuntu as the developers have added a way to automatically

and regularly update some useful information contained in MOTD using cron. To modify how the MOTD is updated, you should install update-motd and read the man page.

You can also make downtimes part of a regular schedule, perhaps to coincide with security audits, software updates, or hardware maintenance.

You should shut down Ubuntu for only a few very specific reasons:

- You are not using the computer, no other users are logged in or expected to need or use the system, such as your personal desktop or laptop computer, and you want to conserve electrical power.
- You need to perform system maintenance that requires any or all system services to be stopped.
- You want to replace integral hardware.

TIP

Do not shut down your computer if you suspect that one or more intruders has infiltrated your system; instead, disconnect the machine from any or all networks and make a backup copy of your hard drives. You might want to also keep the machine running to examine the contents of memory and to examine system logs. Exceptions to this are when the system contains only trivial data files and non-essential services, such as a personal computer that is only used to run a web browser, and when you have no intention of trying to track down what an intruder may have changed, either to repair the damage or to try to catch them using computer forensics, but rather plan to merely wipe everything clean and rebuild or reinstall the system from scratch.

Rebooting the System

You should also use the `shutdown` command to reboot your system. The fastest way to cleanly reboot Linux is to use the `-r` option, and the word `now` or the numeral zero (`0`):

```
matthew@seymour:~$ sudo shutdown -r now
```

or

```
matthew@seymour:~$ sudo shutdown -r 0
```

Both rebooting and shutting down can have dire consequences if performed at the wrong time (such as during backups or critical file transfers, which arouses the ire of your system's users). However, Linux-based operating systems are designed to properly stop active system services in an orderly fashion. Other commands you can use to shut down and reboot Linux are the halt and reboot commands, but the shutdown command is more flexible.

Reading Documentation

Although you learn the basics of using Ubuntu in this book, you need time and practice to master and troubleshoot more complex aspects of the Linux operating system and your distribution. As with any operating system, you can expect to encounter some problems or perplexing questions as you continue to work with Linux. The first place to turn for help with these issues is the documentation included with your system; if you cannot find the information you need there, check Ubuntu's website.

Using Apropros

Linux, like UNIX, is a self-documenting system, with man pages accessible through the `man` command. Linux offers many other helpful commands for accessing its documentation. You can use the `apropos` command—for example, with a keyword such as `partition`—to find commands related to partitioning, like this:

```
matthew@seymour:~$ apropos partition
addpart       (8)    - Simple wrapper around the "add partition" ioctl
all-swaps     (7)    - Event signaling that all swap partitions have been ac...
cfdisk        (8)    - Curses/slang based disk partition table manipulator fo...
delpart       (8)    - Simple wrapper around the "del partition" ioctl
fdisk         (8)    - Partition table manipulator for Linux
gparted       (8)    - Gnome partition editor for manipulating disk partitions.
Mpartition    (1)    - Partition an MSDOS hard disk
Partprobe     (8)    - Inform the OS of partition table changes
Partx         (8)    - Telling the kernel about presence and numbering of on-...
Pvcreate      (8)    - Initialize a disk or partition for use by LVM
Pvresize      (8)    - Resize a disk or partition in use by LVM2
Sfdisk        (8)    - Partition table manipulator for Linux
```

To find a command and its documentation, you can use the `whereis` command. For example, if you are looking for the `fdisk` command, you can do this:

```
$ whereis fdisk
fdisk: /sbin/fdisk /usr/share/man/man8/fdisk.8.gz
```

Using Man Pages

To learn more about a command or program, use the `man` command, followed by the name of the command. Man pages are stored in places like `/usr/share/man` and `/usr/local/share/man`, but you don't need to know that. To read a man page, such as the one for the `rm` command, use the `man` command like this:

```
matthew@seymour:~$ man rm
```

After you press Enter, the less command (a Linux command known as a *pager*) displays the man page. The less command is a text browser you can use to scroll forward and backward (even sideways) through the document to learn more about the command. Type the letter `h` to get help, use the forward slash to enter a search string, or press `q` to quit.

NOTE

Nearly all the hundreds of commands included with Linux each have a man page; however, some do not or may only have simple pages. You may also use the `info` command to read more detailed information about some commands or as a replacement for others. For example, to learn even more about `info` (which has a rather extensive manual page), use the `info` command like this:

```
$ info info
```

Use the arrow keys to navigate through the document and press `q` to quit reading.

The following programs and built-in shell commands are commonly used when working at the command line. These commands are organized by category to help you understand the command's purpose. If you need to find full information for using the command, you can find that information under the command's man page.

- **Managing users and groups**—chage, chfn, chsh, edquota, gpasswd, groupadd, groupdel, groupmod, groups, mkpasswd, newgrp, newusers, passwd, umask, useradd, userdel, usermod
- **Managing files and file systems**—cat, cd, chattr, chmod, chown, compress, cp, dd, fdisk, find, gzip, ln, mkdir, mksfs, mount, mv, rm, rmdir, rpm, sort, swapon, swapoff, tar, touch, umount, uncompress, uniq, unzip, zip
- **Managing running programs**—bg, fg, kill, killall, nice, ps, pstree, renice, top, watch
- **Getting information**—apropos, cal, cat, cmp, date, diff, df, dir, dmesg, du, env, file, free, grep, head, info, last, less, locate, ls, lsattr, man, more, pinfo, ps, pwd, stat, strings, tac, tail, top, uname, uptime, vdir, vmstat, w, wc, whatis, whereis, which, who, whoami
- **Console text editors**—ed, jed, joe, mcedit, nano, red, sed, vim
- **Console Internet and network commands**—bing, elm, ftp, host, hostname, ifconfig, links, lynx, mail, mutt, ncftp, netconfig, netstat, pine, ping, pump, rdate, route, scp, sftp, ssh, tcpdump, traceroute, whois, wire-test

Reference

- https://help.ubuntu.com/community/UsingTheTerminal—The Ubuntu community help page for using the terminal.
- https://help.ubuntu.com/community/LinuxFilesystemTreeOverview—The Ubuntu community help page for and overview of the Linux file system tree.
- https://help.ubuntu.com/community/RootSudo—An Ubuntu community page explaining sudo, the philosophy behind using it by default, and how to use it.
- www.vim.org/—Home page for the `vim` (`vi` clone) editor included with Linux distributions. Check here for updates, bug fixes, and news about this editor.
- www.gnu.org/software/emacs/emacs.html—Home page for the FSF's GNU `emacs` editing environment; you can find additional documentation and links to the source code for the latest version here.
- www.nano-editor.org/—Home page for the GNU `nano` editor environment.

---

800 East 96th Street Indianapolis, Indiana 46240